A PARALLEL ALGORITHM FOR GLOBAL ROUTING

Randall J. Brouwer and Prithviraj Banerjee

Center for Reliable and High Performance Computing

University of Illinois at Urbana-Champaign

1101 W. Springfield Ave.

Urbana, IL 61801

ABSTRACT

A Parallel Hierarchical algorithm for Global Routing (*PHIGURE*) is presented in this paper. The router is based on the work of Burstein and Pelavin [1], but has many extensions for general global routing and parallel execution. Main features of the algorithm include structured hierarchical decomposition into separate independent tasks which are suitable for parallel execution and adaptive simplex solution for adding feedthroughs and adjusting channel heights for row-based layout. In this paper we will be examining closely alternative decomposition methods and the various levels of parallelism available in the algorithm. The algorithm is described and results are presented for a shared-memory multiprocessor implementation.

I. INTRODUCTION

The computational requirements for high quality synthesis and analysis of VLSI designs far outpaces the rapidly growing complexity of VLSI designs. One approach to handle the complexity problem has been to apply parallel processing to certain Computer-Aided Design (CAD) applications[2] because of the advantages of being able to solve *larger problems sizes*, achieve *high quality results*, and affordability of the *low cost multiprocessors*. Along with global routing, research in parallel processing for CAD has included the tasks of floor planning [3], cell placement [4,5,6,7,8], circuit extraction [9], and test generation/fault simulation [10]. This research has demonstrated the wide variety of CAD applications that can be solved with parallel processing.

Acknowledgment: This research was supported by the National Aeronautics and Space Administration under contract NASA NAG 1-613.

In this paper, we present a new parallel algorithm for global routing called *PHIGURE*, a Parallel Hierarchical Global Router. The task of global routing is to take a netlist, a list of pin positions, and a description of the available routing resources and determine the connections and macro paths for each net. Figure 1 shows a simple global routing problem for a chip with pads(P) and standard cells(C) in rows. A global router must make choices beween alternative paths for a net. Some criteria used to evaluate the quality of the routing include: total net length, total chip area, the number of tracks required (row-based routing), the number of feedthroughs used, and the number of vias required. For row-based layout, the output of the global router is used to set up the channels to be routed by a channel router.

Previous research in uniprocessor global routing can be basically divided into these categories: minimum spanning tree solutions [11, 12], maze routing [13], physical analogies [14, 15, 16], and hierarchical routing [1, 17, 18]. Minimum spanning tree solutions model net connections as a spanning graph and try to reduce the graph to a tree while minimizing a cost function. In order to be effective, however, these solutions must handle the net ordering problem. Maze routing solutions apply a line/wave expansion algorithm to route one net at a time. Again, the net ordering problem affects the quality of the results. Physical analogy approaches have modeled the routing problem into the framework of concepts like simulated annealing, attractive and repulsive forces, and electromagnetics. Top-down and Bottom-up hierarchical approaches have also been studied. Other research work on routing has concentrated on combinations of these approaches along with the net rip-up and reroute technique.

In the past, several researchers have proposed parallel approaches to the global routing problem. One approach was to develop a maze routing algorithm suitable for a special purpose hardware routing machine, made up of a 2-D array of microprocessors [19]. Similarly, a maze router was implemented on the AAP-1 2-D array processor [20]. Two other algorithms for maze routing have been developed, specifically for the hypercube [21, 22]. A different approach, developed by Rose for shared-memory multiprocessors [23], determines the best of possible two-bend routes for each two-pin subnet of each net.

Along with the problem of net order dependence, these parallel routing approaches also suffer from routing quality degradation. Thus, since hierarchical routing methods not only route all nets at the same time and incur no routing degradation with parallelism, but also are useful in handling large and complex routing problems, we have developed a parallel top-down hierarchical router.

In Section II of this paper we discuss our global routing model, and the hierarchical routing technique. In Section III we provide an overview of the decomposition strategies applied to subdivide and solve the problem, along with other issues in the algorithm design. Implementation details and parallel processor results for the algorithm are given in Section IV.

II. THE GLOBAL ROUTING FORMULATION

2.1. Global Routing Model

The global routing model we are using is similar to that of Burstein and Pelavin [1]. The entire layout area (including pads) is divided into a two-dimensional array of routing cells. Each routing cell is assigned routing capacity information for each of its four boundaries, based on the physical dimensions of the routing cell. This provides constraints on the number of nets that can be routed through the edges of the routing cell, as in Figure 2.

At each level of the hierarchical decomposition, the current set of routing cells is divided into four regions, forming a two-by-two array of supercells. Thus, each supercell will encompass a sub-region of the layout area. These supercells are further divided at later steps of the decomposition. Each net is cast into one of 15 net types, based on the presence of pins in each of the four supercells. The net types consisting of two or more pins are shown in Figure 3, along with the possible routings for each. Such a formulation was proposed by Burstein and Pelavin [1].

A linear (integer) programming formulation of the problem (LP) is defined such that

For all x, $MAX\ (px)$
subject to $Ax \leq a$ and $Bx = b$,

where x represents the variable space, p represents the objective function, A and a represent the inequality constraints, and B and b represent any equality constraints. In our problem, the set of variables, x_i, $0 \leq i \leq 27$, represent the 28 possible net routings and the set of 15 constraints is based on the available routing capacities and the types of nets being routed. The cost function is designed to minimize interconnection lengths of the nets. The resulting values of the variables x_i represent the number of nets routed in the particular pattern which the variable represents. After a solution to the LP is found, the nets are assigned to the appropriate configuration.

2.2. Estimating Routing Capacities

In the routing capacity model, it is sufficient for each routing cell to maintain capacity information for only two of its four shared edges (for example the top and right edges). Denote the vertical capacity for a routing cell in row r and column c as $v_{r,c}$ (across the top edge), and the horizontal capacity as $h_{r,c}$ (across the right edge). Let L, R, T, and B be the left, right, top, and bottom edges (rows and columns) of the region to be solved. Let X and Y be the locations of the vertical (Y) and horizontal (X) axes respectively of the two-by-two supercell array. Let CAP_i, $i \in A,B,C,D$ represent the capacities of the four axis segments in clockwise order around the two-by-two supercell array, as shown in Figure 4(a). Then

$$CAP_A = \sum_{i=Y}^{T} min(h_{i,X-1}, h_{i,X}, h_{i,X+1})$$

$$CAP_B = \sum_{i=X}^{R} min(v_{Y-1,i}, v_{Y,i}, v_{Y+1,i})$$

$$CAP_C = \sum_{i=B}^{Y} min(h_{i,X-1}, h_{i,X}, h_{i,X+1})$$

$$CAP_D = \sum_{i=L}^{X} min(v_{Y-1,i}, v_{Y,i}, v_{Y+1,i}).$$

This scheme quickly estimates the capacity of the axes with little chance of overestimating by concentrating on the regions closest to the axis. Cases in which the cell capacities are nonuniform near an axis are handled as well. Figure 4(b) illustrates the capacity estimation for the example in Figure 2.

2.3. Feedthrough Insertion and Channel Width Expansion

In row-based layout, feedthroughs must be inserted into the rows to make connections when no built-in feedthroughs or equivalent pins are available when connections must be made from row_i to row_{i+2}. *PHIGURE* handles the problem through the simplex computations. After the problem has been set up, if sufficient routing facilities are available, a solution will be found, else the simplex algorithm will terminate with an infeasible initial problem. By analyzing the simplex state and the given routing problem, adjustments to certain capacities will provide a feasible initial problem for the simplex algorithm. Adjustments to CAP_A and CAP_C are equivalent to increasing the channel width. Adjustments to CAP_B and CAP_D are equivalent to inserting feedthroughs in the row along the X axis. This technique has proven very effective in *PHIGURE*.

2.4. Hierarchical Decomposition

As mentioned earlier, we are applying two-dimensional hierarchical decomposition methods to the global routing problem. At each stage of the hierarchy we divide a larger problem into four smaller sub-problems (divide and conquer). Deciding how to partition the subproblems so that they are independent of each other is very important. One primary decision has to do with how net-crossing locations along the boundaries between the subproblems are determined and locked in place. We have investigated two approaches which are discussed in the following sections.

2.4.1. Maximal Boundary Determination

The first strategy completely determines the net crossing locations by recursively decomposing along the axes of interest down to the routing cell level. This strategy is computationally more costly than the one to be discussed in the next section, but the advantage is that the boundary interface is determined hierarchically as well. Figure 5 shows the first steps in the decomposition for this strategy. The nodes of the graph represent a complete solution of a two-by-two routing instance. The arcs of the graph represent dependencies from child nodes (below) to their parent node (above). In Step 1 and Step 2, the top-most two-by-two solution is followed first by the recursive subdivision and solution of the X axis, down to the level of individual routing cells, and second by the recursive subdivision and solution of the Y axis. After completing these steps, the net crossings have been completely determined and locked into place along both axes of the two-by-two supercell problem, and the four sub-problems for Step 3 are completely independent of each other. This sequence of steps is then recursively repeated until the net crossings across all routing cell edges have been determined. This strategy utilizes the maximum number of two-by-two routing solutions.

2.4.2. Minimal Boundary Determination

Figure 6, shows the first steps in the hierarchical decomposition for this second strategy. The top-most two-by-two problem is solved (Step 1), followed by quick heuristic approximations of the crossings of nets. The four subproblems are then completely independent in Step 2. These steps are repeated recursively until the routing cell level (supercell = routing cell) is reached. This strategy utilizes the fewest two-by-two routing solutions for a hierarchical routing.

The strategy of the minimal determination of the boundary lines is by far the fastest since the number of nodes in the graph (or solutions of two-by-two routing instances) is much less than for the Maximal Boundary Determination strategy; however, there is a trade-off in the expected quality of the solution for computation speed. The routing difficulty comes because without a costly complete analysis, it is extremely hard to best determine exactly where along the boundaries each net should. Some approximations based on the pin locations of each net are used to estimate the crossing, but if the boundaries are not well predicted, the quality of the routing will be severely degraded starting from the top-most two-by-two solution (Step 1). The Maximal strategy takes the extra effort to completely analyze the routing constraints in a hierarchical fashion.

2.5. Task Complexity

In the previous sections, we have discussed some of the basic elements of the two-by-two solution task. These are summarized as follows:

1. Evaluate pin types.
2. Set up linear programming formulation.
3. Solve linear/integer program.
4. Assign routing pattern to each net.
5. Subdivide area for next level of hierarchy.
6. Repeat with child nodes.

THEOREM 1: The complexity of a solution of a *task* is $O(n)$, where n is the number of nets.

Proof: We will show that each subtask is $O(n)$ in the worst case. A circuit is assumed to have $p \approx kn$, where p is the number of pins or net terminals, k is a constant equal to the average number of pins per net, and n is the number of nets in the circuit. Thus p is $O(n)$.

1. To evaluate each pin type requires a search for pins in the current region. This operation is $O(p) \approx O(n)$.
2. Each net is assigned to a specific linear program variable based on the characteristics of the net's pins. This subtask is $O(n)$.
3. The simplex solution of a linear program (with 27 variables, a fixed number independent of the problem size) can be shown to terminate in a finite number of pivots (steps) provided proper pivoting

techniques are used. We are also applying cutting plane methods to convert the linear program solution into an integer solution [24]. Measurements taken show that the average number of pivots in the simplex solution to be less than 6.

4. The current implementation utilizes a very simple assignment algorithm which runs in $O(n)$.

5. Subdivision of the current two-by-two region and setup for the next level of the decomposition can be done in constant time.

Thus, the complexity of a task solution is $O(n)$.

QED

The total complexity of each strategy is the product of the task complexity and the total number of tasks (nodes), which is proportional to the number of routing cells. If M is the total number of routing cells and n is the number of nets, then the computational complexity of *PHIGURE* is $O(nM)$.

2.6. Experimental Results on Task Complexity

In the following figures, the measurements were taken on the Encore Multimax executing the Maximal Strategy on the Primary 1 benchmark. The iteration number refers to the task solution number in a depth-first trace of the execution graph. Figure 7 shows the time taken to setup the net types before the LP solution for each of the task solutions. The average time is 12.9 ms; the standard deviation is 1.2 ms. Figure 8 shows the time taken to solve the given LP problem for each task solution. The average time is 5.7 ms; the standard deviation is 5.3 ms. Figure 9 shows the execution time to assign the net types to a specific configuration for each task solution. The average time is 1.0 ms; the standard deviation is 0.6 ms. Figure 10 shows the total execution time for each task solution. The average time is 19.6 ms; the standard deviation is 5.6 ms.

III. PARALLEL ALGORITHM OVERVIEW

3.1. Exploitation of Coarse-Grained Parallelism

Since the ratio of execution time to synchronization/communication time for the nodes of the execution graph is very large, these tasks are considered to be coarse-grained.

The parallel execution of a binary tree is a well known paradigm, and as we discussed in previous sections, the hierarchical routing execution in *PHIGURE* takes the form of a binary tree in which the nodes of the tree represent the LP set-up, the LP solution, and the net assignments for a single two-by-two routing problem. Furthermore, each node of the tree that is currently being evaluated is completely independent of all other active nodes. The local information for the current sub-problem is derived from its parent node's data structures and global pin location information which is strictly read-only. The solution of the routing subproblem causes the executing process to write the results to a global (shared) output data structure. Since the tasks are spatially independent, there are no contention or race hazards as a process writes out its results.

After writing the results, the process creates two child routing subproblems. One child subproblem is assigned to the first idle and waiting process. The second child subproblem is then executed by the parent itself. If no processes are waiting, the parent will proceed to execute the first subproblem, followed by the second. The number of processes created and initially available for subproblem solution is set equal to the number of processors available to the user.

The routing solution complexity and speedup under parallel execution for both decomposition strategies are estimated in the following sections.

3.1.1. Maximal Boundary Determination

Given R rows and C columns of routing cells, the required number of evaluations to solve the vertical segments of all routing cells in the maximal decomposition strategy is $(R-1)\times(C-1)$. Likewise, the required number of evaluations to solve the horizontal segments is $(C-1)\times(R-1)$. However, one vertical and one horizontal component is solved at each iteration, so the total number of evaluations N_{2x2} is

$$N_{2x2} = (R-1)(C-1).$$

This expression has been verified through actual runs of *PHIGURE*. The estimated execution time for one process is then

$$T_1 = T_{2x2}(R-1)(C-1),$$

where T_{2x2} is the average time to solve a single two-by-two routing problem as a linear function of the number of nets n. The estimated execution time T_P for P processes is equal to the time spent executing

until all P processes are activated plus the time spent in full parallel execution:

$$T_P = (T_{2x2} + T_{sync})\left[2\log_2 P + \log_4 P - 2 + \frac{(R-1)(C-1) - \frac{7P-13}{3}}{P}\right]$$

where T_{sync} is an estimation of the time spent in synchronization. After simplifying the expression, we get

$$T_P = (T_{2x2} + T_{sync})(\frac{(R-1)(C-1)}{P} - \frac{13P-13}{3P} + \frac{5}{2}\log_2 P).$$

The expected speedup is then

$$S_P = \frac{T_1}{T_P} = \frac{T_{2x2}}{(T_{2x2} + T_{sync})} \frac{6P(R-1)(C-1)}{6(R-1)(C-1) - 26P + 26 + 15P\log_2 P}.$$

3.1.2. Minimal Boundary Determination

Again, given R rows and C columns of routing cells, $Z = min(R,C)$, the required number of node tasks to solve is

$$N_{2x2} \leq \sum_{i=0}^{\log_2 Z - 1} 4^i \approx \frac{Z^2 - 1}{3},$$

in which the equality holds for cases when $\log_2 Z$ is an integer. The estimated time for completion for one process is $N_{22} \times T_{22}$. Again, the estimated execution time for P processes is equal to the time spent executing until all P processes are activated plus the time spent in full parallel execution:

$$T_P = (T_{2x2} + T_{sync})\left[\log_4 P + \frac{\frac{Z^2-1}{3} - \frac{P-1}{3}}{P}\right]$$

After simplifying the expression, we get

$$T_P = (T_{2x2} + T_{sync})(\frac{Z^2 - P}{3P} + \frac{1}{2}\log_2 P).$$

The expected speedup is then

$$S_P = \frac{T_1}{T_P} = \frac{T_{2x2}}{(T_{2x2} + T_{sync})} \frac{2P(Z^2-1)}{2Z^2 - 2P + 3P\log_2 P}.$$

Figure 11 provides a graphical look at the previous set of equations assuming that $\frac{T_{sync}}{T_{2x2}} = 0.1$. Included in the plot is an estimate of process efficiency (useful time/ total time) and its effect on the possible speedup. The current implementation provides dynamic task scheduling based on process availability. Thus, due to task granularity, there will be times when a process waits idle for a new task to be generated. As the number of processes increases, the process efficiency is expected to decrease.

3.2. Exploitation of Fine-Grained Parallelism

There are three specific subtasks which can be executed in parallel at a fine-grained level. First, during the LP setup, the type for each net of the current two-by-two problem is determined. Since each net is independent, the nets may be divided between available processes and evaluated in parallel. Second, the exchange operations required to solve the linear/integer program may also be divided between available processes for parallel execution. Finally, the assignment of nets could also be done in parallel, based on specific net types. Each of these areas of parallelism are orthogonal to each other.

However, since the amount of parallelism available at the task level is so great, the exploitation of parallelism at the fine-grain level would not provide significant improvement. Only during the startup phase of the execution tree will there be specific processes idle. Figure 12 shows the percentage of the number of two-by-two solutions in the startup phase to the total number of two-by-two solutions for routing problems with $R = C = Z$ and $P = 16$. As is clear from the figure, the part of the execution in large sized problems for which fine-grain parallelism can be useful is extremely small. Furthermore, parallelism of the simplex solution would not be effective since the average number of pivoting operations for solution is less than 6. Therefore, we determined that is was unnecessary to implement these tasks at such a fine-grain level.

IV. IMPLEMENTATION AND RESULTS

PHIGURE was implemented using approximately 5000 lines of C code on an eight processor Encore Multimax 510 (shared memory multiprocessor), in which each processor is a National 50310 CPU. Experiments were performed on a few of the placement and routing benchmarks from the MCNC Workshop on Placement and Routing, along with a number of other circuits. Testing was done for a single process, two processes, four processes, and eight processes.

Table 1 compares the routing results of *PHIGURE* to actual runs of the TimberWolf 5.4 global router (TW) [11], and some of the recently published results for the UTMC router (UT) [11], a router by Cong and Preas (CP) [12], and Locusroute (LR) [23], This table shows that *PHIGURE* performs well within the range of some recently published routers. Table 2 compares the uniprocess runtimes for the TimberWolf 5.4 router with those of *PHIGURE*. These measurements were taken on a Sun 3/110 workstation.

Table 3 shows the results for two of the Placement and Routing Workshop benchmark circuits and three other standard cell circuits. For each circuit, the table gives the number of tracks used as estimated by the maximum channel density and the average execution times in seconds (real time, including process creation) for one, two, four, and eight processes using the Minimal and Maximal decomposition strategies. Cell placements for all of the circuits were performed by TimberWolf 5.4. As is clear from the table, there is no degradation in routing quality when going from a single process to many processes, and very good speedups were achieved (>6 for 8 processes). Since the hierarchical decomposition creates a large number of jobs after the first few steps, our algorithm is scalable for a large numbers of processes.

V. CONCLUSIONS

In this paper we have presented a new parallel global router, *PHIGURE*, which applies hierarchical routing and decomposition techniques to create independent subproblems which can be evaluated in parallel. Results were presented which compare two strategies for decomposing the routing problem and show that high quality routings are attainable for one strategy. Most importantly, the routing quality is not degraded by decomposing in parallel. We are currently investigating improved task scheduling schemes, improved net assignment techniques, incorporating fine grain parallelism, implementation on a message-passing multiprocessor, extensions to a combined place and route algorithm, and strategies for adaptive rerouting of problem nets.

REFERENCES

[1] M. Burstein and R. Pelavin, "Hierarchical Wire Routing," *IEEE Trans. CAD*, vol. CAD-2, No. 4, pp. 223-234, Oct. 1983.

[2] P. Banerjee, "The Use of Parallel Processing in VLSI Computer-Aided Design Applications," ICCAD-88 Tutorial, also Tech. Report No. CSG-104, Coordinated Science Laboratory, Univ. of Illinois, Urbana, IL, May 1988.

[3] R. Jayaraman and R. A. Rutenbar, "Floorplanning by Annealing on a Hypercube Multiprocessor," *Proc. Int. Conf. Computer-Aided Design*, pp. 346-349, Nov. 1987.

[4] J. Sargent and P. Banerjee, "A Parallel Row-Based Algorithm for Standard Cell Placement with Integrated Error Control," *Proc. 26th Design Automation Conf.*, pp. 590-593, Jun. 1989.

[5] S. A. Kravitz and R. A. Rutenbar, "Placement by Simulated Annealing on a Multiprocessor," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, No 4, pp. 534-549, Jun. 1987.

[6] A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli, "A Parallel Simulated Annealing Algorithm for the Placement of Macro-Cells," *Proc. Int. Conf. Computer-Aided Design*, pp. 30-33, Nov. 1986.

[7] J. S. Rose, D. R. Blythe, W. M. Snelgrove, and Z. G. Vranesic, "Fast, High Quality VLSI Placement on a MIMD Multiprocessor," *Proc. Int. Conf. Computer-Aided Design*, pp. 42-45, Nov. 1986.

[8] R. M. Kling and P. Banerjee, "ESP: Placement by Simulated Evolution," *IEEE Transactions Computer-Aided Design*, vol. CAD-8, No 2, pp. 245-256, March 1989.

[9] K. P. Belkhale and P. Banerjee, "PACE: A Parallel VLSI Circuit Extractor on the Intel Hypercube Multiprocessor," *Proc. Int. Conf. Computer-Aided Design*, pp. 326-329, Nov. 1988.

[10] S. Patil and P. Banerjee, "Fault Partitioning Issues in an Integrated Parallel Test Generation/Fault Simulation Environment," *Proc. Int. Test Conf.*, pp. 718-726, Aug. 1989.

[11] K. W. Lee and C. Sechen, "A New Global Router for Row-Based Layout," *Proc. Int. Conf. Computer-Aided Design*, pp. 180-183, Nov. 1988.

[12] J. Cong and B. Preas, "A New Algorithm for Standard Cell Global Routing," *Proc. Int. Conf. Computer-Aided Design*, pp. 176-179, Nov. 1988.

[13] R. Nair, "A Simple Yet Effective Technique for Global Wiring," *IEEE Trans. CAD*, vol. CAD-6, No. 2, pp. 165-172, Mar. 1987.

[14] M. P. Vecchi and S. Kirkpatrick, "Global wiring by simulated annealing," *IEEE Trans. Computers*, vol. C-7, pp. 215-222, Oct. 1983.

[15] N. Hasan and C. L. Liu, "A Force-Directed Global Router," *Proc. Stanford Conference on Advanced Research in VLSI*, pp. 135-150, 1987.

[16] C. D. Hechtman and J. J. Lewandowski, "A Flux Directed Approach to a Wire Routing Problem," *IEEE VLSI Technical Bulletin*, vol. 4, No. 3/4, pp. 124-138, Sep./Dec. 1989.

[17] M. Marek-Sadowska, "Global Router for Gate Array," *Proc. Int. Conf. Computer Design*, pp. 332-337, Oct. 1984.

[18] W. K. Luk, D. T. Tang, and C. K. Wong, "Hierarchical Global Wiring for Custom Chip Design," *Proc. 23rd Design Automation Conference*, pp. 481-489, June 1986.

[19] R. Nair, S. J. Hong, S. Liles, and R. Villani, "Global Wiring on a Wire Routing Machine," *Proc. 19th Design Automation Conference*, pp. 224-231, Jun. 1982.

[20] T. Watanabe, H. Kitazawa, and Y. Sugiyama, "A Parallel Adaptable Routing Algorithm and its Implementation on a Two-Dimensional Array Processor," *IEEE Transactions Computer-Aided Design*, vol. CAD-6, No 2, pp. 241-250, March 1987.

[21] O. A. Olukotun and T. N. Mudge, "A Preliminary Investigation into Parallel Routing on a Hypercube Computer," *Proc. 24th Design Automation Conference*, pp. 814-820, June 1987.

[22] Y. Won and S. Sahni, "Maze Routing On A Hypercube Multiprocessor Computer," *Proc. Int. Conf. on Parallel Processing*, pp. 630-637, Aug. 1987.

[23] Jonathan Rose, "LocusRoute: A Parallel Global Router for Standard Cells," *Proc. 25th Design Automation Conference*, pp. 189-195, June 1988.

[24] R. S. Garfinkel and G. L. Nemhauser, *Integer Programming*. New York, NY: John Wiley and Sons, Inc., 1972. pp. 154-165.

List of Figures

List of Tables

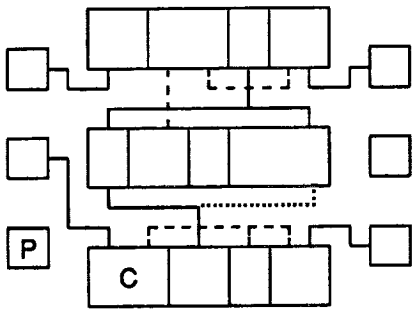

Figure 1. Example Global Routing Problem.

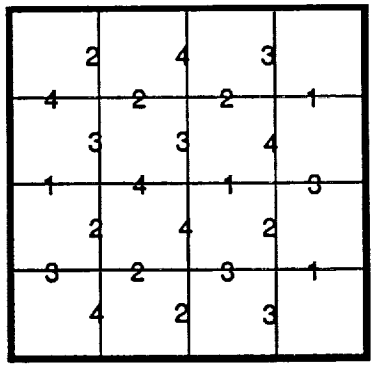

Figure 2. Routing Cell Model.

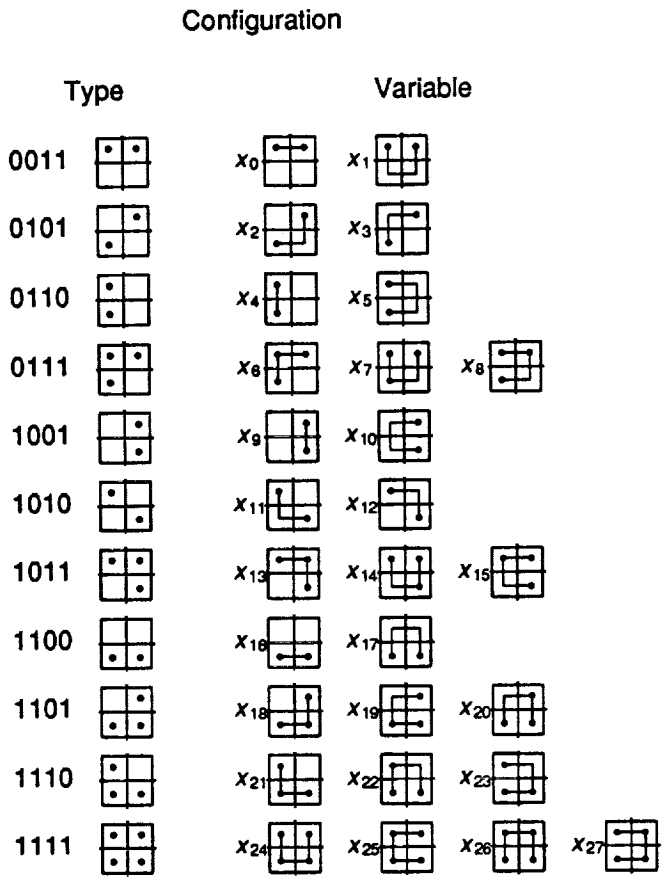

Figure 3. Net Types and Possible Routings.

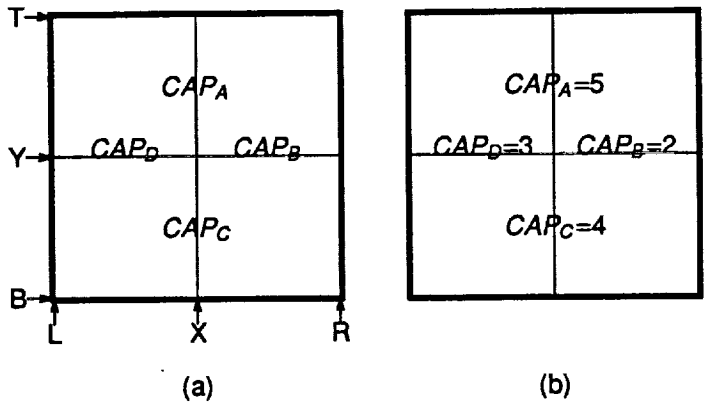

Figure 4. (a) Axes Capacities of 2x2 Supercell (b) Example.

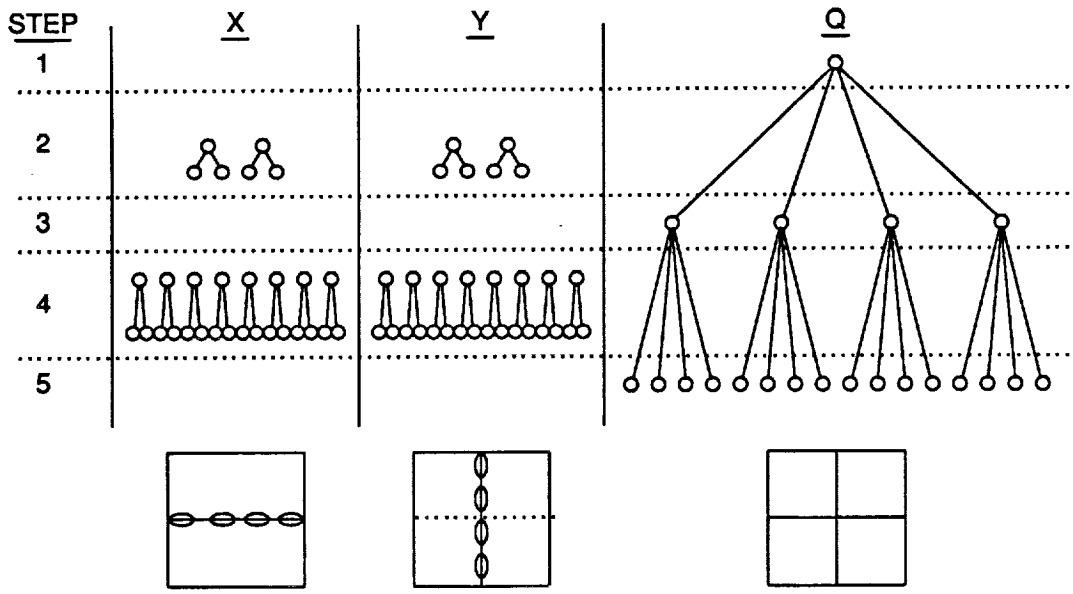

Figure 5. Maximal Boundary Determination.

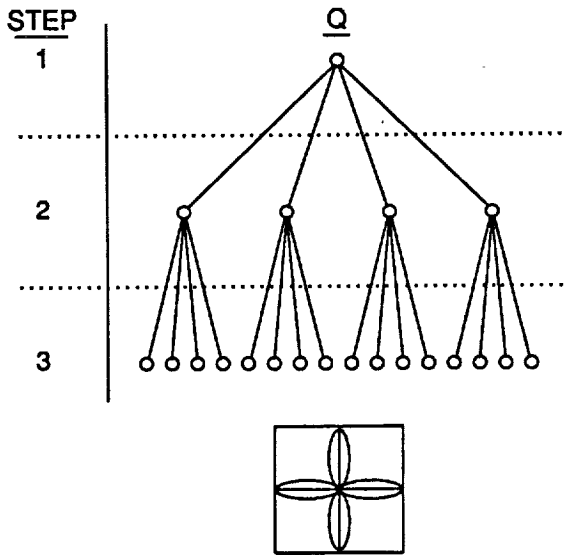

Figure 6. Minimal Boundary Determination.

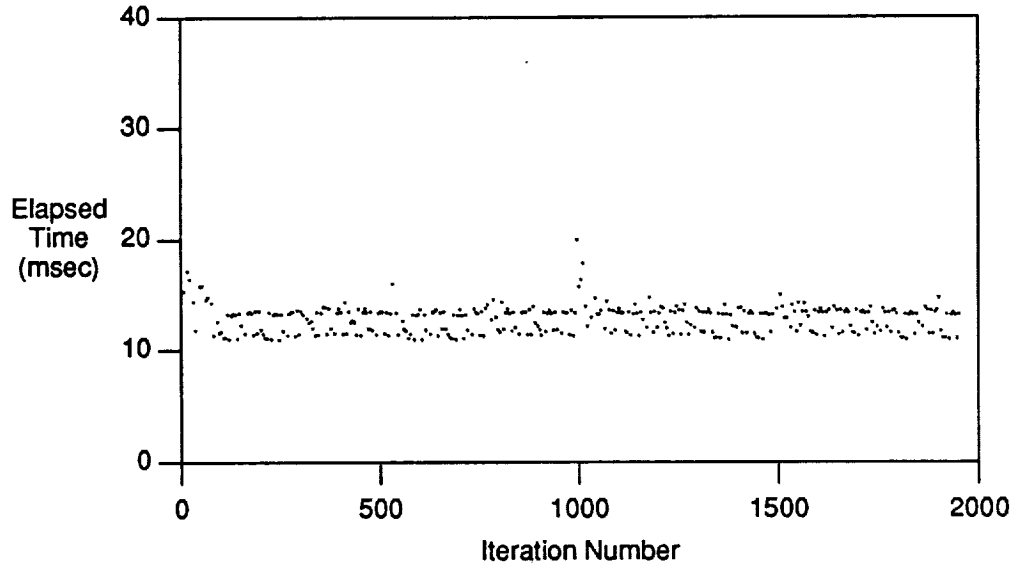

Figure 7. Net Setup Time vs. Iteration Number.

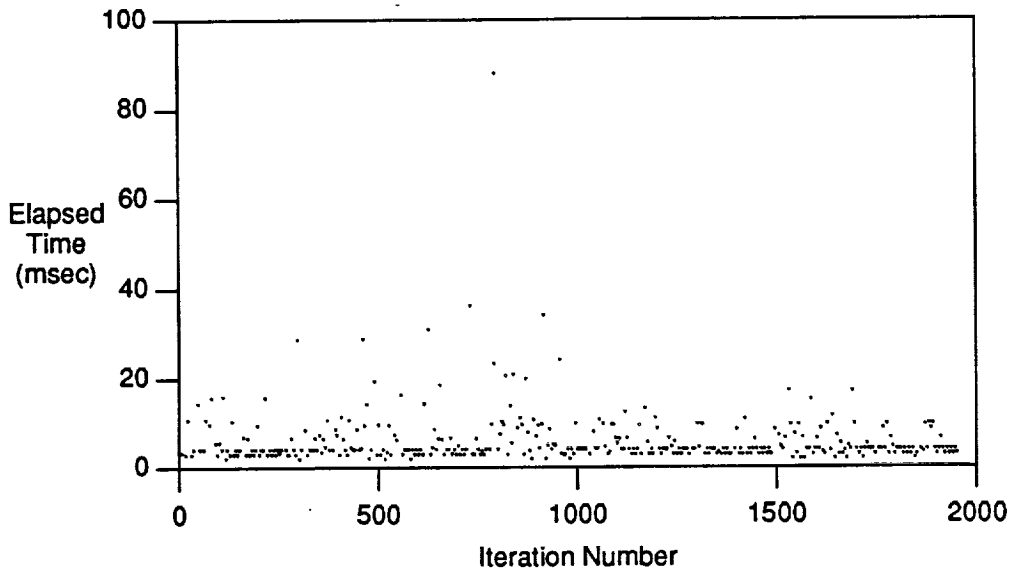

Figure 8. LP Solution Time vs. Iteration Number.

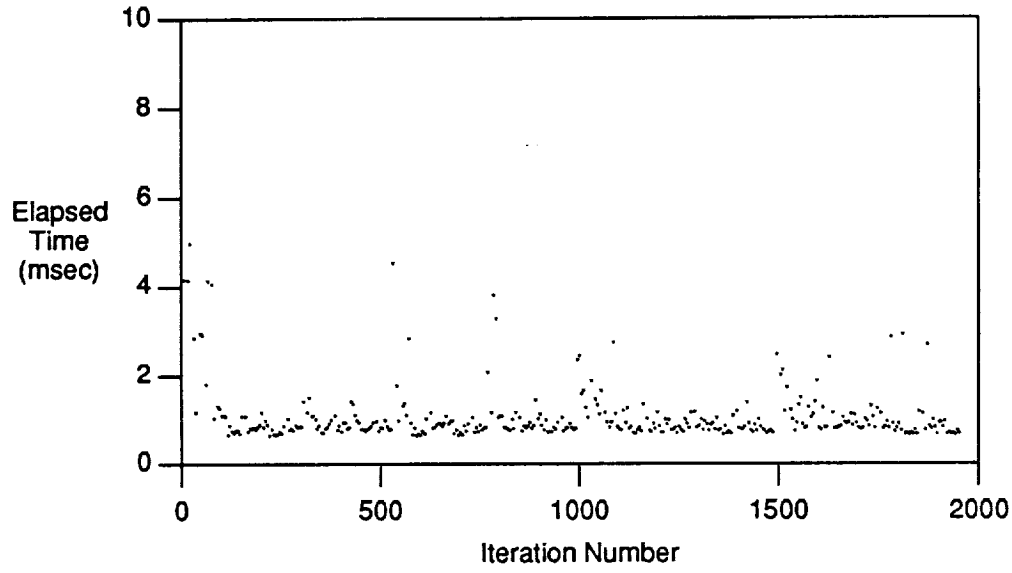

Figure 9. Net Assignment Time vs. Iteration Number.

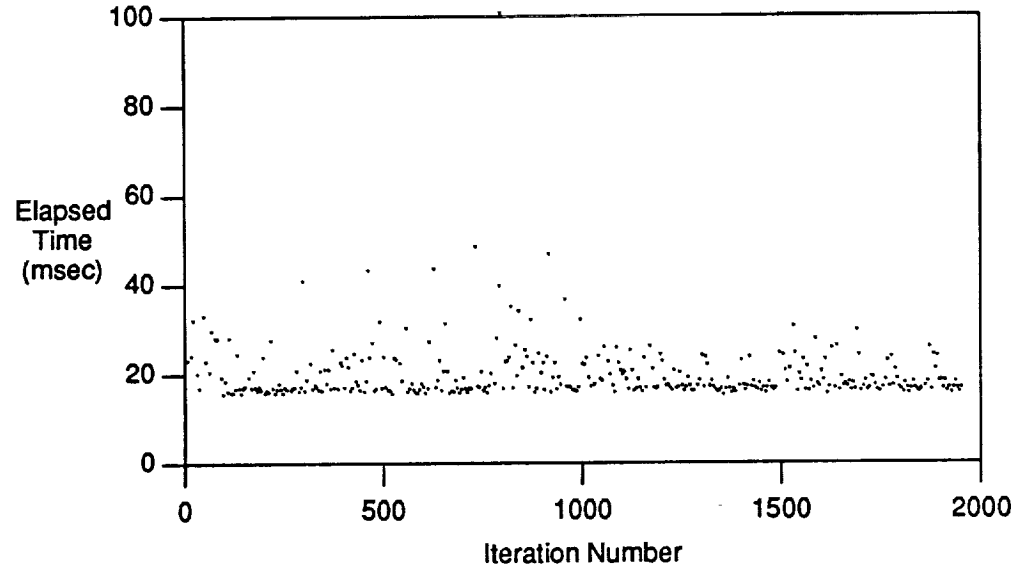

Figure 10. Total Time vs. Iteration Number.

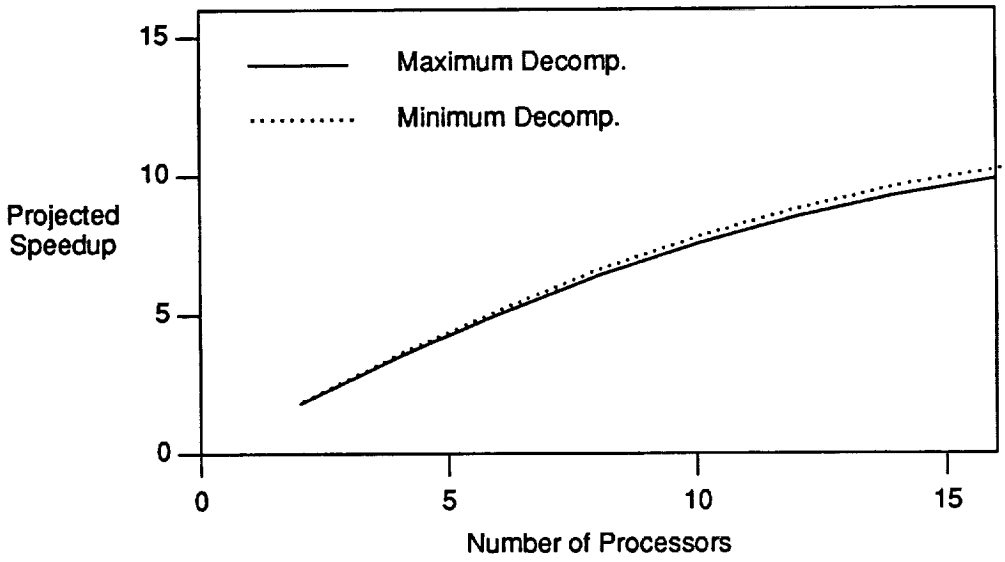

Figure 11. Plot of Projected Speedup vs. Number of Processes.

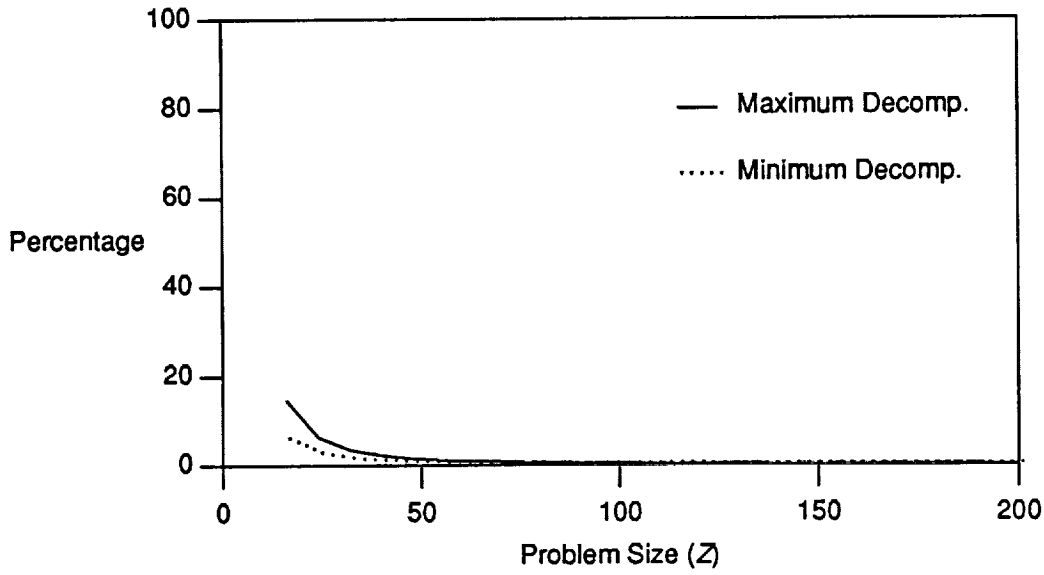

Figure 12. Percentage of Tasks in Startup Phase.

Table 1. Routing Quality Comparison.

	Number of Trks				
Circuit	*PHIGURE*	TW5.4	UTMC	CongPreas	LocusRoute
Primary1	177	163	177	190	262
Primary2	404	432	447	449	563

Table 2. Uniprocessor Runtime Comparison on a Sun 3/110.

	Runtime (secs)	
Circuit	TimberWolf 5.4	*PHIGURE*
Primary1SC	812	214
Primary2SC	3883	1116

Table 3. Parallel Algorithm Results.

Circuit (Nets)	P	Min Decomp			Max Decomp		
		Trks	Time(sec)	SpdUp	Trks	Time(sec)	SpdUp
Primary1 (1185)	1	348	33	1.0	177	66	1.0
	2	348	17	1.9	177	34	1.9
	4	348	9	3.7	177	21	3.1
	8	348	6	5.5	177	14	4.7
Primary2 (3710)	1	817	187	1.0	404	257	1.0
	2	817	97	1.9	404	131	1.9
	4	817	52	3.6	404	74	3.5
	8	817	30	6.2	404	42	6.1
Circuit X1 (1979)	1	641	189	1.0	416	202	1.0
	2	641	92	2.0	416	101	2.0
	4	641	47	4.0	416	59	3.4
	8	641	29	6.5	416	36	5.6
Circuit X2 (3013)	1	709	254	1.0	515	284	1.0
	2	709	139	1.8	515	145	1.9
	4	709	74	3.4	515	79	3.6
	8	709	44	5.7	515	44	6.5
Circuit X3 (3258)	1	742	192	1.0	625	389	1.0
	2	742	97	1.9	625	198	1.9
	4	742	52	3.7	625	106	3.7
	8	742	30	6.4	625	67	5.8